Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Redescubriendo la plataforma robótica experimental MASHI

MEMORIA

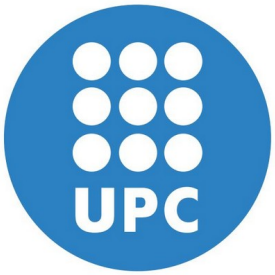
Autor: Joaquín Cortés Fuentes
Director: Cecílio Angulo Bahón

Convocatòria: Enero 2017

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

REDESCUBRIENDO LA PLATAFORMA ROBÓTICA EXPERIMENTAL MASHI

JOAQUÍN CORTÉS FUENTES



Grau en Enginyeria en Tecnologies Industrials
Enginyeria de Sistemes, Automàtica i Informàtica Industrial (ESAII)
Universitat Politècnica de Catalunya
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB)

Enero 2017

SUPERVISORES:
Cecilio Angulo Bahón
ENTREGADO:
Enero 2017

Resumen

La plataforma MASHI nació como un experimento para estudiar la interacción social entre seres humanos y un robot, en colaboración entre la UPC (Universitat Politècnica de Catalunya, Barcelona, España) y la ESPOL (Escuela Superior Politénica del Litoral, Guayaquil, Ecuador). Para ello, se creó el robot implementando el hardware y software necesario y se realizaron pruebas en un centro cultural, obteniéndose los resultados comentados en [10]. Sin embargo, una vez finalizado el trabajo, Dennys Pailacho (el creador de MASHI) trasladó la cabeza y los brazos del robot original a la ESPOL, quedando en la UPC la base del cuerpo. Así, en este proyecto lo que se busca es reconstruir la plataforma original, entendiendo su funcionamiento y documentando el proceso para facilitar el posterior trabajo que pueda hacerse en futuros proyectos. Una vez hecho, se proponen una serie de mejoras para optimizar la plataforma y enriquecer su funcionalidad. En concreto, en este proyecto lo que se busca es simplificar el proceso de movimiento por parte del operador, añadiendo un sistema de navegación autónoma para facilitar en gran medida la acción de moverse del robot y comenzar un *framework* para posteriores proyectos relacionados.

Abstract

The MASHI platform was born as an experiment to study the social interaction between human beings and a robot, in collaboration between the UPC (Universitat Politècnica de Catalunya, Barcelona, España) and the ESPOL (Escuela Superior Politénica del Litoral, Guayaquil, Ecuador). To do it, the robot was created implementing the necessary hardware and software and several tests were performed in a cultural center, obtaining the results shown in [10]. But nevertheless, once the project was finished, Dennys Pailacho (MASHI's creator) moved the head and the arms from the original robot to the ESPOL, leaving in the UPC only the base from the body, So, what is wanted in this project is to reconstruct the original platform, understanding its behaviour and documenting the process in order to ease future works. Once done, some improvements are proposed in order to optimize the platform and enrich its functionality. Specifically, in this project what is wanted is to simplify the process of movement by the operator, adding an autonomous navigation system in order to ease in a large degree the robot's move action and to start a framework for later similar projects.

Índice general

Índice de figuras

1 Objetivos y alcance

Uno de los objetivos principales de este proyecto es reconstruir la plataforma MASHI para devolverle su estado original, así como sus funcionalidades principales (movimiento teleoperado y transmisión en directo de vídeo y sonido). Se irá documentando el proceso, así como una explicación de los diferentes subsistemas de la plataforma para facilitar el trabajo futuro. Para esta parte, se ha trabajado conjuntamente con Xavier Rodríguez [12], cuyo proyecto es común con este en este objetivo. Con ello se busca crear un *framework* de trabajo para poder continuar y mejorar la plataforma con proyectos futuros, facilitando la accesibilidad al trabajo hecho y empezando líneas de trabajo para optimizar e investigar en profundidad, las cuales quedan fuera del alcance de este proyecto.

Una vez recuperada la plataforma original, se propusieron las siguientes mejoras con objeto de optimizar el robot, mejorar sus funcionalidades y aumentar la autonomía:

- Implementar un sistema de navegación autónoma, para que el operador simplemente tenga que indicar el destino en un mapa del entorno, en lugar de mover al robot manualmente mediante teleoperación.
- Optimización del hardware interno de la plataforma, sustituyendo el PC interno que tenía por un microprocesador (en concreto, una *Raspberry Pi*).

El primer objetivo es el que se trabajará en este proyecto, mientras que el segundo se realiza en [12].

En cuanto al alcance del proyecto, la reconstrucción del MASHI original es un objetivo a realizar de forma prioritaria, ya que si no se cumple no puede continuarse el proyecto. Una vez hecho, se proceden a implementar las mejoras. Al realizarse de forma simultánea, ambas mejoras se implementan de forma independiente, de manera que el no progreso de una no interfiera en la otra. En cuanto a la mejora de navegación, se buscará que la plataforma MASHI sea capaz de moverse por un entorno conocido de manera autónoma, esquivando los obstáculos fijos, haciendo que el operador simplemente tenga que indicarle el punto final de su trayectoria. Este entorno se supondrá interior y de un tamaño relativamente pequeño, además de simple (sin una gran cantidad de obstáculos o formas inusuales). De esta manera se enriquecerá la funcionalidad de la plataforma, se simplificará la labor del operador, y se abrirá la puerta a futuras mejoras y su implementación de manera más simple (como por ejemplo, la programación de una ruta).

2 Descripción plataforma MASHI

El robot MASHI consiste en una serie de elementos interconectados con la finalidad de realizar una tarea social en su caso original (para estudiar el comportamiento humano frente a un robot de estas características). A continuación se describirán los diferentes subsistemas de los cuales consiste el robot y como están conectados entre sí con la finalidad de dotarlo de movimiento y transmisión de imagen y sonido en tiempo real, a través de un operador que maneja al robot a distancia mediante comunicación por Internet. Se enfatiza una explicación general del funcionamiento de los diferentes subsistemas, sin entrar en detalles sobre el funcionamiento de cada componente específico (así, no se explica el funcionamiento concreto de los microprocesadores, de los códigos utilizados, del sistema de control y otros aspectos).

2.1. Descripción física. Hardware y conexionado

La plataforma MASHI consiste en una base de madera de 22,6 cm de radio y 1 cm de grosor, situada a 10 cm de altura sobre el suelo, sobre la cual están instaladas las ruedas y los motores de movimiento del cuerpo. Además, sobre esta base reposan también los diferentes elementos eléctricos y electrónicos que permiten el movimiento de traslación, así como el ordenador que permite la comunicación con el operador. En la Figura 1 puede verse el estado de la base original.

Figura 1: Base de MASHI. Se observa la batería que alimenta al robot.

Para el sistema de movimiento se usa el kit *Motor Mount and Wheel Kit with Position Controller* de Parallax, el cual incluye las ruedas, los motores y un *encoder* capaz de leer velocidad, posición y otras funcionalidades. Además, se usa un controlador de motor HB-25, también de la empresa Parallax, el cual permite un mejor control de los motores mediante modulación de ancho de

pulsos *PWM* [1]. Junto a estas dos ruedas motoras se utilizan dos ruedas no controladas para darle estabilidad el movimiento. Estos elementos se conectan a un microcontrolador *Arduino Mega 2560*, el cual a su vez se conecta al PC del robot. El microcontrolador se encarga de recibir la orden del PC para moverse en una dirección concreta y dar las señales adecuadas a los controladores de los motores mediante un puerto serie para efectuar el movimiento deseado y leer información sensorial de los motores (en concreto, velocidad y posición angular del eje de cada motor). Estos motores van conectados a una batería de 12 V para alimentación. Se usa además un interruptor que permite desconectar los motores en cualquier momento.

Figura 2: MASHI recubierto con telas

El robot tiene una altura de 110 cm hasta la cabeza con el fin de dar una estética humanoide, aunque esta altura puede modificarse. Tiene dos secciones cubiertas con tela para ocultar su interior y dar un aspecto más agradable, como puede verse en la Figura 2.

La cabeza está hecha con piezas de plástico diseñadas expresamente para el robot e impresas en 3D. Dentro de la cabeza se alberga una pantalla de 7 pulgadas de la marca *SainSmart*, donde se representa una cara mediante la cual se intenta transmitir confort a quien vea al robot, dando expresiones humanas. Se dispone también de una cámara, un altavoz y un micrófono que permiten la transmisión en tiempo real de imagen y sonido, así como la emisión de audio. Estos elementos van conectados directamente al ordenador del robot. Además, la cabeza dispone de 3 servomotores *Dynamixel AX-12A* para efectuar movimientos de rotación en los 3 ejes. Estos motores son controlados de una manera diferente a los de la base, utilizando un microcontrolador *Robotis CM 9.04A* para tal finalidad. Estos servomotores pueden controlarse directamente con la controladora, sin necesidad de utilizar técnicas como modulación de pulsos. Este tipo de motores están diseñados para conectarse entre ellos y realizar una única conexión a la controladora y otra a la alimentación, simplificando todo el conexionado. En la Figura 3 puede verse una imagen de la cabeza original sin la carcasa exterior.

MASHI dispone también de 2 brazos, uno de ellos móvil con 2 grados de libertad, con una cámara en la posición de la mano, que le permite hacer fotografías y subirlas a Twitter en tiempo real; y el otro con un solo grado de libertad, pudiendo mover el antebrazo para dar un saludo. Estos brazos están unidos al cuerpo mediante un perfil de aluminio. Las articulaciones

1 La modulación por ancho de pulsos (*PWM*, por sus siglas en inglés) es una técnica en la que se modifica una señal con la finalidad, que esta entregue menos energía a la carga y poder modificar así la potencia que recibe, y en el caso de los motores, poder controlar su velocidad.

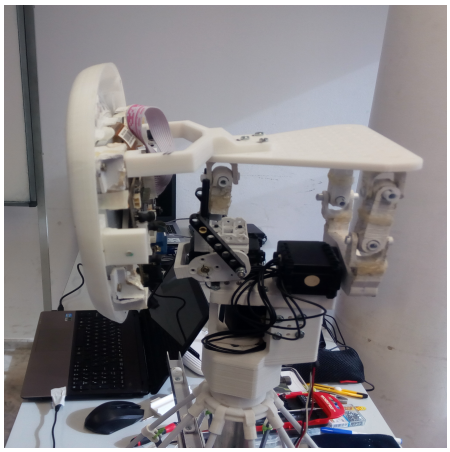
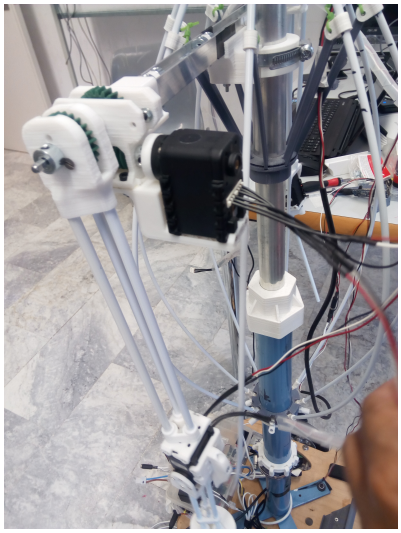


Figura 3: Cabeza de MASHI original sin carcasa

están formadas por piezas impresas en 3D y motores *Dynamixel* (del mismo tipo que los de la cabeza), mientras que las uniones entre estos elementos se hacen con varillas de PVC de 5 mm de diámetro. Los brazos van recubiertos de espuma protectora. En la Figura 4 y Figura 5 pueden verse imágenes de los brazos.



(a) Brazo izquierdo original (hombro)

(b) Brazo izquierdo original (antebrazo)

Figura 4: Brazo izquierdo original de MASHI, sin espuma protectora

El PC del MASHI se encarga de recibir las órdenes adecuadas del operador a través de una conexión de Internet y transmitirlas a los diferentes elementos (en concreto, a los microcontroladores, al monitor, altavoz y ambas cámaras) para que actúen como se desee. También transmite la información que recibe por las cámaras y por el micrófono para su visualización por el operador. Se ha utilizado un PC ASUS X555L para este proyecto, pero cualquier otro serviría mientras este configurado para leer correctamente los dispositivos y ejecutar el software adecuado.

Figura 5: Brazo derecho original de MASHI, sin espuma protectora

Las características de todos los elementos utilizados pueden consultarse en el Anexo A de este proyecto.

En la Figura 6 puede verse el esquema eléctrico de los componentes de la base. Tanto el diodo como la resistencia se utilizan para proteger al microcontrolador de posibles tensiones que puedan dañarlo. No se ha hecho un esquema eléctrico de la cabeza al ser todas las conexiones directas (los motores de rotación se conectan directamente a la placa Robotis, que su vez va conectada al PC).

2.2. Descripción del software. Control y transmisión

Para el control del robot MASHI se utilizan varias herramientas para permitir el correcto movimiento y la transmisión de datos a distancia. En la Figura 7 puede verse un esquema general de la arquitectura de comunicación.

En primer lugar se tiene el PC central del MASHI, el cual permite la comunicación con el operador a través de Internet siempre y cuando ambos estén conectados a la misma red. Este dispositivo hace de servidor web utilizando *node.js*[2] para que el operador pueda transmitir y recibir fácilmente datos, ya sean de audio o vídeo. El PC del operador ha de abrir simplemente una página web alojada en MASHI donde se transmite vídeo en directo y puede realizar movimiento, emitir sonido o cambiar la salida del monitor, tal como puede verse en la Figura 8. Para ello, se envían los datos adecuados a

2 *node.js* es un entorno de ejecución para JavaScript, usado ampliamente para aplicaciones web por su facilidad de uso y por permitir conexionado múltiple.

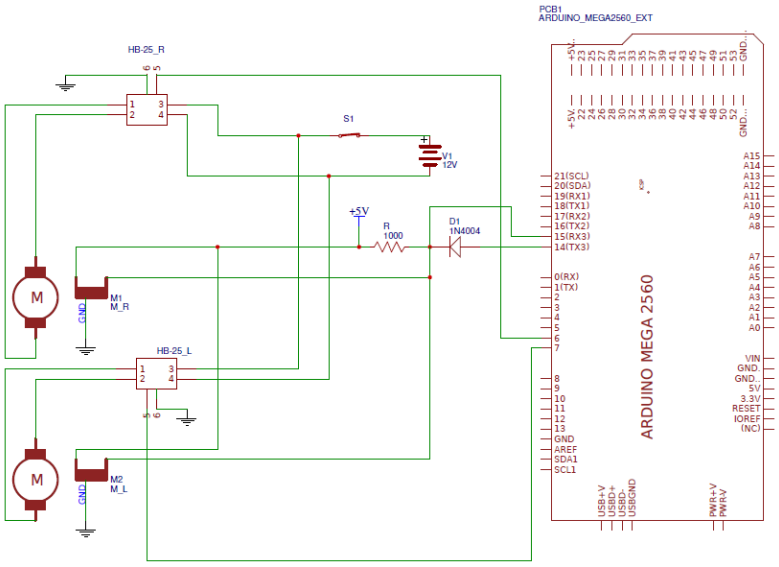


Figura 6: Esquema eléctrico de la base de MASHI

Figura 7: Esquema de la comunicación entre operador y MASHI

través de Internet al servidor alojado en MASHI, donde este se encarga de dar las órdenes adecuadas a los diversos dispositivos conectados, los cuales se han detallado antes. Este servidor está programado a base de funciones de escucha, de manera que cada vez que se reciben datos se ejecutan unas funciones u otras dependiendo del dato recibido.

Para la transmisión de movimiento de la plataforma, el operador simplemente elige la dirección de movimiento mediante las teclas adecuadas, y desde su PC se envía la orden al PC de MASHI, el cual a su vez envía la orden al microprocesador *Arduino*. Este lee la señal recibida y el código se encarga del control de los motores a través de sus respectivos controladores, limitando la velocidad máxima, realizando el giro adecuado de cada motor en cada instante e incluyendo un control PID digital para garantizar la estabilidad (utilizando los *encoders* de los motores para leer la velocidad). El control se realiza mediante *PWM*: con los controladores HB-25 se modula la alimentación que reciben los motores, cambiando así la velocidad de giro de estos. Todo el código utilizado en este proyecto puede verse en [11].

También se controla el movimiento de la cabeza, la cual tiene 3 grados de libertad para realizar movimientos de rotación. Para ello, el operador envía la orden y el PC de MASHI controla el microprocesador *Robotis*, el código

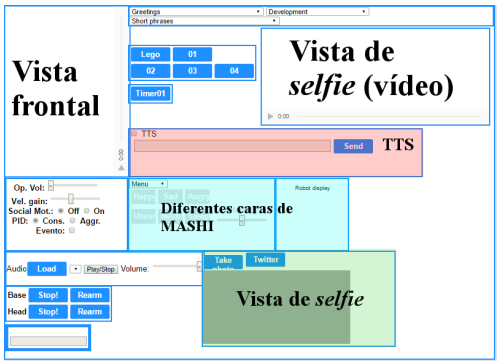


Figura 8: Web del operador para controlar a MASHI. Se ven las zonas donde se efectuan las diferentes acciones.

del cual controla adecuadamente los servomotores *Dynamixel* para realizar el movimiento deseado.

Los demás componentes utilizados se controlan directamente a través de los puertos USB, sin necesidad de un microprocesador adicional. El robot es capaz de comunicación verbal, *hablando* a través de un módulo de síntesis de voz de *Google*, el cual convierte texto en voz. El operador simplemente teclea el texto en la web a través del campo de TTS (*Text-To-Speech*), el cual se envía al PC de MASHI y este lo convierte en voz, emitiendo el sonido por el altavoz.

Figura 9: Cara de MASHI

En el PC de MASHI también hay abierta una página web a través de la cual se muestran las expresiones faciales del robot, controladas por el operador. Esta página se muestra a través del monitor de MASHI. En la Figura 9 puede verse un ejemplo de ello.

La transmisión de imagen se realiza con la cámara incorporada en la cabeza, la cual transmite vídeo en tiempo real, captando la imagen, enviándola al PC de MASHI y este al PC del operador. La cámara del brazo es capaz de hacer fotos y subirla a Twitter en tiempo real, gracias a otro módulo del servidor dedicado a ello.

En el anexo C de este proyecto puede verse un pequeño manual para inicializar y controlar la plataforma MASHI.

3 Proceso de reconstrucción

Al inicio del proyecto, la plataforma MASHI se encontraba en un estado incompleto. Había que reconstruir toda la cabeza y los brazos, tanto la estructura como los componentes internos necesarios. Para ello, se dispuso de los modelos en 3D de todas las piezas que formaban las diferentes partes, así como su ensamblaje, y se conformaron a través de impresión 3D en plástico PLA. Este proceso permite una gran flexibilidad en cuanto a la forma de las piezas y da una resistencia estructural suficientemente buena para el tipo de robot que es el MASHI. Además, permite una fabricación rápida y barata en comparación a otros métodos de conformado. En la Figura 10 puede verse un ejemplo de piezas hechas con impresión 3D. También se dispuso de una lista de todos los componentes necesarios, los cuales pueden verse en el Anexo A de este proyecto.

Figura 10: Ejemplo de piezas impresas en 3D

Una vez fabricadas todas las piezas, se procede a realizar el montaje. Las piezas ya están diseñadas para facilitar el montaje de manera sencilla, así como para albergar los motores y otros componentes, mediante tornillos y roscas ordinarias. Otras piezas, como las articulaciones universales, van pegadas para mantener una estructura fija y sólida. En la Figura 11 se ve la cabeza ya reconstruida.

Figura 11: Cabeza reconstruida de MASHI, sin carcasa

Para los brazos se realizó el mismo procedimiento que para la cabeza, imprimiendo las diferentes piezas y uniéndolas con tornillos y roscas. Los brazos están formados también por tubos de PVC de 5 mm de diámetro, que hacen la función de brazo y antebrazo (mientras que las otras piezas hacen de articulaciones). Los dos brazos son idénticos, salvo que uno alberga los motores necesarios para realizar el movimiento y la cámara al final del brazo, mientras que el otro tiene un solo motor. Los brazos van unidos al cuerpo mediante un perfil de aluminio de 15x15x400 mm. En la Figura 12 se ven los brazos ya reconstruidos.

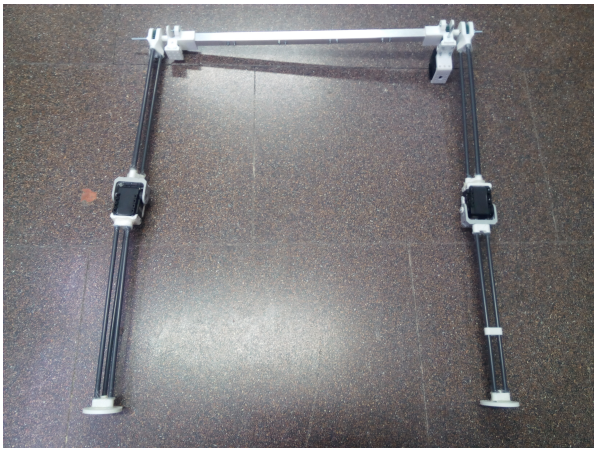


Figura 12: Brazos reconstruidos de MASHI

La base ya estaba montada, así que no se ha modificado su estructura ni se ha reconstruido ninguna parte.

Una vez hecha la parte estructural, se dispuso a realizar las conexiones necesarias para que la plataforma MASHI funcione correctamente. En la base las conexiones ya estaban hechas y simplemente hubo que conectar el *Arduino* al PC. Para la cabeza y los brazos, se conectaron los motores al controlador *Robotis*, y este al PC, juntamente con los altavoces, cámaras, micrófono y pantalla. Este PC también fue configurado para poder establecer una conexión con sus diferentes componentes y con el PC del operador: en concreto, se instalaron los softwares de Arduino y *node.js* (el cual permite el establecimiento de un servidor web y manejar los diferentes dispositivos conectados), las librerías necesarias de WebRTC[1] y otras relacionadas, y los controladores de todos los dispositivos. Los microcontroladores se alimentan con la propia conexión USB al PC, mientras que la pantalla y los motores se alimentan con una batería de 12 V montada en la base (también existe una toma de alimentación para conectar la plataforma a la red eléctrica, aunque con esta función se restringe la movilidad a la longitud del cable).

En la Figura 13 se ve todo el robot MASHI reconstruido, sin la carcasa de la cabeza ni las conexiones hechas.

1 WebRTC es un conjunto de librerías y métodos que permiten a las aplicaciones web realizar funciones de vídeo, sonido y uso compartido de archivos P2P.

Figura 13: MASHI reconstruido (sin carcasa ni conexiones)

4 Propuestas de mejora

Una vez reconstruida la plataforma MASHI original, el siguiente objetivo fue implementar las mejores propuestas en el Capítulo 1. Como se comentó en ese mismo aparatado, las dos propuestas se desarrollan en paralelo, de manera que se hacen sin tener en cuenta a la otra. En este apartado se desarrolla la idea general sobre la mejora a trabajar en este proyecto, y más adelante se discute su implementación y los requisitos necesarios para ello.

Lo que se busca al mejorar el sistema de navegación es añadir una nueva funcionalidad de movimiento, sin sustituir a la teleoperación manual (la cual puede seguir siendo usada si el operador así lo requiere), para que el robot encuentre y siga correctamente una ruta entre su posición inicial y su objetivo. Para ello, existen diversas técnicas, como el seguimiento de un objetivo mediante balizas u otros elementos externos (como líneas u otros elementos). Sin embargo, estas técnicas, aunque simples, requieren de elementos externos y añaden severas restricciones en cuanto al espacio de movimiento disponible para el robot. Para evitar este problema, existen los algoritmos de *path-finding*, el funcionamiento de los cuales varía según el propio algoritmo, pero cuya idea general consiste en calcular la trayectoria óptima en un mapa conocido. De este modo, se puede dotar al robot de un sistema de navegación potente y con libertad en el espacio de movimiento, sin restringirlo severamente ni depender a priori de elementos externos para ello. A continuación se discuten algunos de los algoritmos existentes y utilizados actualmente, para finalmente escoger el más adecuado.

4.1. Algoritmos *path-finding*

La idea general de los algoritmos es dividir el mapa del entorno en celdas, cada una con un coste asociado, y calcular el camino (formado por celdas conectadas entre ellas) que una los puntos de inicio y fin que tenga el menor coste asociado. Para ello, cada uno de los algoritmos usa diferentes técnicas, con sus respectivas ventajas e inconvenientes:

- En primer lugar tenemos el algoritmo A*, unos de los más utilizados debido a su simplicidad y su eficiencia computacional. Este algoritmo calcula el coste del punto inicial hasta el actual y utiliza una heurística para calcular el camino *aparentemente más corto* hasta el final; luego elige el camino más corto para avanzar al siguiente nodo, y vuelve a comenzar una nueva iteración del algoritmo hasta llegar al final. La principal ventaja de este algoritmo es su simplicidad de implementación y su bajo coste de ejecución. Sin embargo, su principal inconveniente es la memoria utilizada y el tiempo de ejecución, que pueden crecer de manera exponencial en el peor de los casos (dependiendo de la heurística utilizada y del mapa en sí). Existen diversas variaciones de este algoritmo, como el IDA*, LPA*, SMA*, y muchos otros.

- Otro de los algoritmos es el D*, el cual funciona de una manera similar al A*, pero con la diferencia que el D* permite navegar en terreno desconocido, haciendo suposiciones sobre él (como que se encuentre libre de obstáculos) y cambiando los valores de coste según va encontrando nuevo terreno, de manera que, aunque tiene un mayor coste computacional, permite obtener una ruta de coste mínimo en un entorno con elementos desconocidos o cambiantes. Sin embargo, hoy en día se utiliza una variación de este algoritmo, conocido como D*-lite, el cual tiene la misma filosofía pero funciona de una manera matemáticamente diferente (detallada en [6]). Este algoritmo permite reducir el coste computacional requerido, dando los mismos resultados que el D*.

- Existe otro tipo de algoritmo, el PRM (*Probabilistic Road Maps*), el cual tiene un funcionamiento diferente a los citados anteriormente. En este algoritmo, en lugar de calcular directamente la ruta de menor coste, se usan muestras aleatorias de los puntos del mapa y se conectan entre sí los más cercanos, hasta que el grafo de puntos es suficientemente denso como para finalmente buscar el camino más corto entre los puntos inicial y final. Este algoritmo siempre converge a la ruta más corta (o a la no-existencia de solución si es el caso). Por la naturaleza probabilística del método, cada vez que se ejecute el algoritmo se trazarán rutas diferentes, por lo que en la misma situación, el mismo algoritmo puede dar dos soluciones diferentes. Este tipo de algoritmos funciona muy bien con robots con muchos grados de libertad, ya que es mucho más eficiente un enfoque probabilístico que utilizar un algoritmo determinista para tantos grados de libertad. Su principal inconveniente es que, dada la naturaleza probabilística, existe la posibilidad de que no se encuentre un camino, o que sea necesario un muestreo grande para asegurarse de ello (con la consiguiente necesidad de memoria).

De entre estos algoritmos, se descartan el PRM ya que su coste de memoria y tiempo puede ser elevado si se quiere un muestreo denso (y por consiguiente, una probabilidad alta de encontrar un camino óptimo). Además, es un tipo de algoritmo más adecuado para otro tipo de aplicaciones. Los algoritmos D* también se descartan, ya que aunque eficaces, están destinados principalmente a mapas desconocidos y su complejidad de implementación y costes de memoria y tiempo son más elevados que los derivados del A*. Así, para este proyecto se usará un algoritmo de la familia del A*, y en el siguiente capítulo se verá cual es la implementación más eficaz para este proyecto.

Así, un robot dotado de un sistema de *path-finding* puede encontrar el camino óptimo en un entorno conocido. Sin embargo, para implementar un sistema de navegación, se requieren otros elementos adicionales de los que no dispone MASHI, y los cuales se detallan a continuación:

- Sistema de mapa: el robot necesita una manera de poder interpretar un mapa dado por el operador[1], para poder posicionarse sobre él y poder hacer los cálculos necesarios para ejecutar el algoritmo de *path-finding*.

1 Aunque podría tratarse el problema como un problema de SLAM (*Simultaneous Locations And Mapping*) y así poder trabajar en entornos desconocidos, no se ve necesario para este

- Sistema de posicionamiento: para poder seguir la trayectoria deseada, MASHI necesita poder localizarse sobre el mapa, sabiendo su posición y orientación absoluta en todo momento.
- Sistema de movimiento automático: se requiere un sistema de movimiento para tal de poder darle la trayectoria calculada y que el mismo robot siga el camino adecuado, dando las órdenes necesarias en cada momento para seguir el movimiento deseado.

A continuación se discuten estos subsistemas en detalle, buscando diferentes implementaciones y escogiendo una para este proyecto.

4.2. Mapeado

Los algoritmos de *path-finding* necesitan un mapa dividido en nodos para tal de poder utilizarse. Así, se requiere un mapa del entorno discreto, donde cada nodo contenga información sobre su coste de camino (el cual también refleja si el nodo es alcanzable de alguna manera o por el contrario se trata de un obstáculo). Existen varias maneras de representar un mapa de esta manera, algunas más sofisticadas que otras (como el uso de mapas en forma de imágenes, ya sean de tamaño fijo o en formato vectorial), pero para este proyecto se ha escogido una representación bastante simple, representando el mapa como una matriz binaria de ceros y unos. Cada posición de la matriz representa las coordenadas del punto del mapa, y el elemento correspondiente (0 o 1) representa si esa celda es alcanzable o no (es decir, si es un obstáculo o no). Este método permite introducir el mapa de una manera directa y con la precisión que se quiera, aunque tiene la desventaja que el operador ha de introducir manualmente el mapa en forma de matriz. Para obtener esta representación del mapa, existen diversos softwares que permiten transformar una imagen en una matriz numérica (por ejemplo, *MATLAB* incluye una herramienta de procesado de imagen, la cual permite obtener una matriz numérica a partir de una imagen que represente el mapa).

4.3. Localización

El robot necesita una manera de saber su posición y orientación en coordenadas globales (tomando como referencia el entorno en sí, que se supone estático), ya que sin ello es imposible seguir la trayectoria de manera adecuada. Existen muchas maneras de obtener la posición de un objeto en un entorno, pero cada una de ellas depende de varios factores, principalmente de si el entorno es interior o exterior, de la precisión que se busque, y de la tecnología utilizada. A continuación se revisan varios de estos métodos para tener una comparativa:

- Odometría: también conocido como *dead-reckoning*, consiste en calcular la posición y orientación del objeto en cuestión mediante el conteo

proyecto este grado de libertad, ya que la complejidad asociada a este tipo de problemas esta más allá de lo que pretende este proyecto, y se ha supuesto que el robot trabajará en un entorno conocido, pequeño y fijo.

de las vueltas que dan las ruedas, las cuales son las encargadas del movimiento. Sin embargo, resulta más fácil calcular la velocidad del objeto y su velocidad angular a partir de las velocidades angulares de las ruedas, y luego integrar este resultado para obtener la variación de posición y orientación del objeto. Este método resulta fácil de implementar, pero tiene el inconveniente de resultar poco preciso a la larga (el resultado tiende a divergir), además de basarse en suposiciones que pueden no resultar realistas según el caso (como el no deslizamiento de las ruedas).

- IMU (*Inertial Measurement Unit*): estos dispositivos consisten en un conjunto de acelerómetros y giroscopios que se encargan de medir la aceleración y la velocidad angular del objeto en su base local. Estos resultados pueden luego integrarse y combinarse adecuadamente para dar como resultado la variación de posición y orientación. Estos dispositivos son fácilmente implementables en cualquier sistema, pero su principal inconveniente es el error, que no está acotado y tiende a crecer a medida que avanza el tiempo (además, un pequeño error en la medición puede ocasionar un error no despreciable en el resultado final, debido a que el error se integra también).

- Triangulación y trilateración: estos métodos permiten conocer la posición de un objeto a través de parámetros geométricos relacionados con puntos fijos llamados nodos o balizas. En la triangulación, se usan los ángulos que forman las diversas líneas que unen las balizas y el objeto móvil, mientras que en la trilateración se usan las distancias entre el objeto y las balizas. Para obtener estos parámetros, existen diversidad de métodos, entre ellos el GPS (útil para entornos exteriores) o el uso de balizas por radiofrecuencias, ultrasonidos, etc. A su vez, estos métodos tienen varias maneras de calcular las variables de interés (pudiendo usar como base de cálculo el tiempo de llegada de la señal del eco, la potencia recibida de la baliza, etc.). Estos métodos pueden ser muy precisos, pero el inconveniente principal es la necesidad de añadir elementos externos para la localización (normalmente más del mínimo necesario para evitar los problemas de atenuación de la señal). Además, según la tecnología usada (radiofrecuencias, ultrasonidos, etc.), se necesita una precisión temporal que no todos los sistemas pueden alcanzar (pudiendo llegar al orden de los nanosegundos). Finalmente, en entornos interiores, se encuentran problemas con los ecos de la señal, pérdidas y otros problemas de atenuación e interferencias que dificultan el paso de la señal.

Además de estos métodos, existen muchos otros como el uso de escáners por láser en 3D o el uso de cámaras para analizar la imagen y obtener la posición en función de algún parámetro. Sin embargo, estos métodos no se discuten en este proyecto dada su complejidad de implementación.

La solución más adoptada para obtener un sistema de posicionamiento preciso consiste en utilizar varias de las alternativas citadas anteriormente y combinar los resultados mediante un filtro adecuado. De esta manera, se intenta compensar los errores de una fuente con los resultados de la otra.

Para la fusión de los datos sensoriales, existen diversos algoritmos de filtraje que permiten obtener una buena estimación de los datos. Uno de los filtros más utilizados hoy en día es el filtro de Kalman, la teoría del cual puede verse detallada en [14]. Este filtro permite fusionar datos de diversos sensores y reducir el ruido de manera considerable. Este algoritmo permite obtener mediciones muy precisas, pero requiere de un modelo matemático completo del sistema, y su implementación es compleja, ya que ha de modelarse el filtro para cada sistema concreto.

Una alternativa simple al filtro de Kalman es el filtro complementario, el cual consiste en pasar una de las mediciones por un filtro HPL (*High Pass Filter*) y el otro por uno LPL (*Low Pass Filter*), de manera que la suma de las funciones de transferencia de ambos filtros sea siempre 1, y sumar los resultados. Puede verse la teoría de funcionamiento en [8].

Para este proyecto, se usará un sistema simple de odometría. Se utiliza este método ya que la misma plataforma MASHI está equipada con un sistema que permite medir la velocidad angular de las ruedas, y no es necesario añadir hardware adicional. El principal inconveniente de este método es que la precisión que se alcanza no es del todo precisa, y el error es acumulativo, pero para una primera aproximación al problema es suficiente. Una vez decidido como el robot MASHI tendrá su sistema de navegación, en el siguiente capítulo se procede a discutir como será su implementación particular.

5 Implementación sistema de navegación

Una vez sabido cómo hará MASHI para interpretar un mapa, localizarse sobre él y calcular un camino a seguir, se procede a implementar este sistema. Para ello se añadirán nuevas funcionalidades a los programas ya existentes, realizando las modificaciones necesarias para tal de poder realizar las tareas descritas. Uno de los objetivos es que estos subsistemas sean lo más independientes entre ellos (excepto el subsistema de seguimiento de trayectoria, el cual depende del resto).

5.1. Implementación de localización

Para implementar el sistema de localización basado en odometría, primero ha de obtenerse una expresión para obtener la posición del punto de interés (en este caso, el punto central de la base) en función de las velocidades angulares de las ruedas. Como se comentó en el capítulo anterior, es más fácil trabajar con magnitudes de velocidad que con magnitudes de posición directamente, y luego integrar estos resultados. Para ello, se parte del modelo de la base como se ve en la Figura 14. Al estar el movimiento limitado al plano, se simplifican todos los cálculos. En la Figura 15 también se ve una rueda en vista lateral.

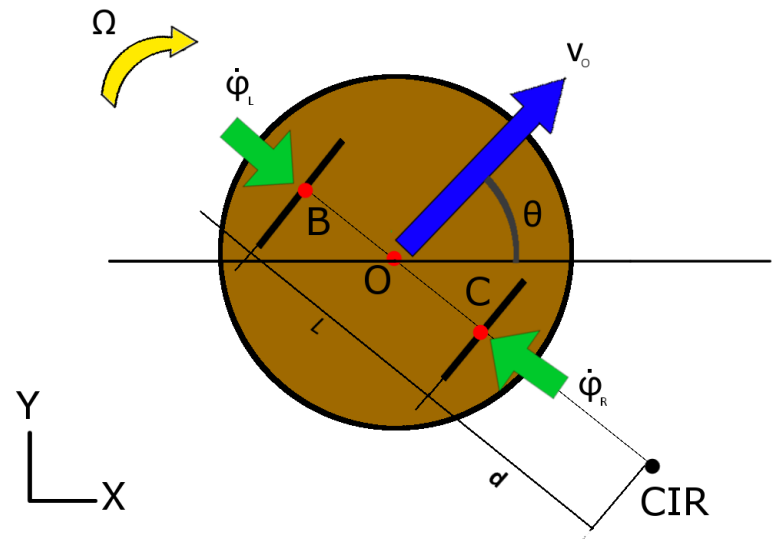


Figura 14: Modelo cinemático de la base

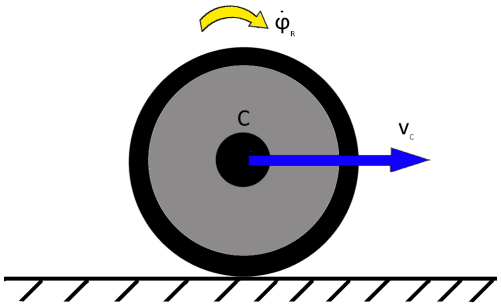


Figura 15: Modelo cinemático de la rueda

De la Figura 15 se puede observar que la velocidad de los puntos *B* y *C* [1] (situado en el mismo plano que el punto central de interés, *O*) es directamente proporcional a la velocidad angular:

$$v(C) = R\dot{\varphi}_R \tag{1}$$
$$v(B) = R\dot{\varphi}_L \tag{2}$$

siendo *R* el radio de la rueda y $\dot{\varphi}_r$ es la velocidad angular de la rueda correspondiente, $r = \{R, L\}$. Ambas ruedas son simétricas, así que puede efectuarse el cálculo con cualquiera de ellas: en este caso, se usa la rueda derecha, aunque el resultado sería el mismo usando la otra rueda. La velocidad siempre tendrá la misma dirección en la referencia solidaria a la base del robot, así que puede usarse la magnitud escalar en lugar de la vectorial.

Utilizando la siguiente expresión, se puede calcular la velocidad de O:

$$\vec{v}(O) = \vec{v}(C) + \vec{\Omega} x \overrightarrow{CO} \tag{3}$$

siendo $\overrightarrow{CO}$ el vector entre *C* y *O*, y $\vec{\Omega}$ la velocidad angular de la base, la cual se desconoce a priori, pero puede calcularse fácilmente empleando esta misma expresión con el *CIR*[2] y con ambas ruedas:

$$\vec{v}(CIR) = 0 = \vec{v}(C) + \vec{\Omega} x \vec{d} \tag{4}$$
$$\vec{v}(CIR) = 0 = \vec{v}(B) + \vec{\Omega} x (\vec{d} + \vec{L}) \tag{5}$$

siendo *d* la distancia entre el punto *C* y el *CIR* y *L* la distancia entre las ruedas, como se ve en la figura XX.

De las Ecuaciones 1,2, 4, 5, y teniendo en cuenta la Figura 14 (donde pueden verse las direcciones de las diversas magnitudes mencionadas) se pue-

1 Si no especifica lo contrario, todas las ecuaciones están expresadas en referencia del robot. Los módulos han de coincidir en ambas bases.

2 El Centro Instantáneo de Rotación (*CIR*) es el punto respecto al cual el sólido gira en cada instante. Tiene la propiedad de tener velocidad nula, y es un concepto fundamental en la cinemática de sólidos rígidos.

den deducir las siguientes ecuaciones escalares para obtener Ω y d, teniendo en cuenta que el movimiento es 2D:

$$\begin{cases} R\dot{\varphi}_R + d\Omega = 0 \\ R\dot{\varphi}_L + (d + L)\Omega = 0 \end{cases} \tag{6}$$

por lo que pueden aislarse los valores de d y de Ω:

$$d = \frac{L\dot{\varphi}_R}{(\dot{\varphi}_L - \dot{\varphi}_R)} \tag{7}$$

$$\Omega = \frac{-R\dot{\varphi}_R}{d} = \frac{R}{L}(\dot{\varphi}_R - \dot{\varphi}_L) \tag{8}$$

Puede verse que d puede ser negativo, indicando que el *CIR* se desplaza hacia el lado contrario. También se ve que existe una singularidad cuando las dos ruedas tengan la misma velocidad angular (cuando hay únicamente movimiento lineal de traslación, sin rotación), pero este hecho puede evitarse añadiendo una excepción a la hora de implementar esta ecuación.

Aplicando las Ecuaciones 1, 3 (en forma escalar), se obtiene la velocidad del punto central O:

$$v(O) = R\dot{\varphi}_R + (d + 0{,}5L)\Omega \tag{9}$$

donde hay que sustituir los valores correspondientes de d y de Ω usando las Ecuaciones 7, 8.

Esta velocidad esta expresada en la referencia solidaria a la base, pero interesa obtenerla en referencia global para tal de obtener la posición absoluta. Para ello, viendo la figura 14 se puede deducir que:

$$\vec{v}(O)_{|ABS} = v(O)_{|REL} \begin{Bmatrix} \cos(\theta) \\ \sin(\theta) \end{Bmatrix} \tag{10}$$

donde los subíndices *ABS* y *REL* hacen referencia a las bases global y del robot, respectivamete, y $v(O)_{REL}$ se corresponde con el valor definido en Ecuación 9. El valor θ, como se ve en la Figura 14, hace referencia al ángulo de orientación de la base respecto el eje X global. Para obtener este ángulo, ha de integrarse Ω para obtener la variación del ángulo de orientación $\Delta\theta$. La integración se realiza de forma numérica, y tomando la velocidad angular constante en el intervalo de tiempo (el cual es suficientemente pequeño para tomar esta simplificación como correcta), se obtiene:

$$\Delta\theta = \frac{\pi}{2} - \Omega\Delta t \tag{11}$$

Sabiendo la orientación inicial, puede saberse la orientación en cada intervalo de tiempo haciendo la suma acumulativa de $\Delta\theta$.

Se realiza el mismo procedimiento con la velocidad de O, con las mismas suposiciones para obtener la variación de posición del punto en coordenadas absolutas:

$$\begin{Bmatrix} \Delta x \\ \Delta y \end{Bmatrix} = v(O)_{|REL} \begin{Bmatrix} \cos(\theta) \\ \sin(\theta) \end{Bmatrix} \Delta t \tag{12}$$

De esta manera se obtiene la variación de posición absoluta del robot. Sabiendo la posición inicial y haciendo la suma acumulativa de los valores de Δx y Δy, se obtiene la posición absoluta, simplemente usando la Ecuación 12 y sustituyendo los valores adecuados usando las Ecuaciones 11, 9, 8, 7.

Estas expresiones se implementan en el código de *Arduino*, dentro del bloque *loop()*, con una ligera modificación: se añade un bloque de excepción si la diferencia entre las velocidades de los motores es menor que un valor de tolerancia ϵ[3], de manera que si los valores son prácticamente iguales, se tome el movimiento como lineal ($\Omega = 0$) y así evitar la singularidad en la Ecuación 7. El código puede consultarse en [11]. Los valores de posición y orientación se envían al PC de MASHI mediante comunicación por puerto serie, desde donde se leen con *node.js* y se procesan de manera adecuada. Se almacenan como una lista de tres valores, con las coordenadas X e Y en referencia global y el ángulo θ de orientación.

5.2. Implementación del mapa

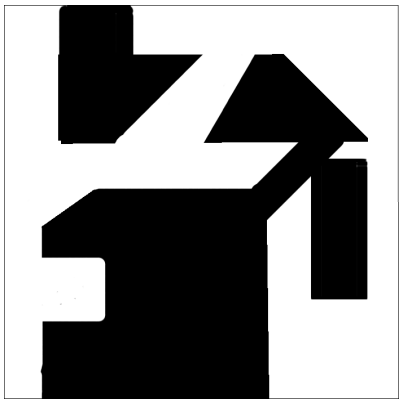
Para representar el mapa, como se comentó en el Capítulo 4, se utiliza una representación de 0 y 1 en forma de matriz. Cada elemento de esta matriz representa una zona cuadrada del entorno real, donde se ha escogido que cada casilla del mapa sea una zona de 1 cm. De esta manera, se tiene una resolución del mapa suficientemente buena para ejecutar los algoritmos de *path-finding* (una resolución menor no daría mejores resultados, ya que la precisión del sistema de odometría no permitiría una mejor ejecución).

Para no introducir manualmente el mapa directamente en el código (ya que sería un trabajo costoso y propenso a error humano), primero se dibuja el mapa con una herramienta de dibujo externa, procurando usar únicamente dos colores (preferiblemente blanco para la zona navegable y negro para el resto) y respetando la proporción de $1\text{px} = 1\text{cm}$. Seguidamente, se usa el software *MATLAB* para convertir la imagen en una matriz binaria. Esta matriz puede pegarse en una hoja de cálculo de *Excel* y leerse mediante la librería *xlsx* de *node.js*. Luego se trata para que tenga el formato adecuado de matriz de JavaScript. En la Figura 16 se ve un ejemplo de mapa en este formato.

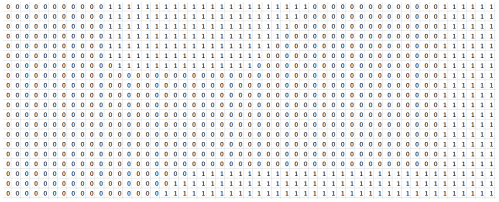
5.3. Implementación de *path-finding*

Como se comentó en el Capítulo 4, se usaría un algoritmo de la familia de los A* en la implementación. Para ello, se aprovecha la librería *PathFinding.js* de *node.js*, desarrollada por Xueqiao Xu, y cuya documentación puede consultarse en [15]. Gracias a este paquete, se añaden funciones potentes de *path-finding* al entorno de *node.js*, incluyendo una variedad de algoritmos de la familia de los A*. Entre ellos, caben destacar los siguientes: A*, IDA*, Búsqueda en anchura, Algoritmo de Dijkstra y JPS (*Jump Point Search*. También se incluyen otros algoritmos que no aseguran el camino óptimo y versiones

3 Se toma un valor de tolerancia ya que por la propia naturaleza de los sensores, el valor puede fluctuar y dar valores ligeramente diferentes aun cuando las condiciones sean las mismas. Además, las operaciones matemáticas se hacen a cabo usando números representados por coma flotante, los cuales tienen precisión limitada.



(a) Formato gráfico.



(b) Formato binario (detalle).

Figura 16: Ejemplo de mapa en formato gráfico y binario. En este caso, el mapa es de 700x700 cm.

bidireccionales de algunos algoritmos, pero no se discuten en este proyecto. De entre estos algoritmos, se descarta el IDA* ya que tiene un tiempo de ejecución más elevado que el resto [4], ya es posible que se revisen varias veces algunos de los nodos, ralentizando así el proceso, aunque tiene la ventaja de utilizar menos memoria que el A*: sin embargo, teniendo en cuenta los entornos para los cuales está pensado MASHI y la precisión que tienen los mapas, los requerimientos de memoria no son tan elevados como para necesitar un ahorro en este tema. También se descarta la búsqueda en anchura ya que requiere revisar todos los nodos del mapa, incluso aquellos más alejados que intuitivamente se podrían descartar desde un principio. Así mismo, el algoritmo de Dijkstra requiere una revisión de todos los nodos (ya que este algoritmo calcula la distancia más corta del nodo inicial a *todos* los otros nodos del mapa). Finalmente, se decide utilizar el JPS, ya que aunque el A* sigue siendo una buena alternativa, el JPS permite reducir notablemente el tiempo de ejecución al realizar ciertas simplificaciones, entre ellas la de coste uniforme para todos los nodos alcanzables, como se discute en [4]. Para comprobarlo, se realiza una prueba para encontrar el camino óptimo en un mapa de prueba como el de la Figura 16. Se ejecutan los algoritmos A* y JPS, contando el tiempo de ejecución en ambos casos. Las pruebas se han realizado en el mismo PC de MASHI, cuyas características se especificaron en el Capítulo 2. Para el algoritmo A*, se obtuvo un tiempo de ejecución de 33 ms, mientras que con el JPS se obtuvo 17 ms. De esta manera, se comprueba el JPS encuentra el mejor camino en la mitad de tiempo que el A*.

Con esta librería, el camino calculado se obtiene como una lista de puntos conectados, los cuales forman el camino. Así, se obtienen todos los puntos por los cuales ha de pasar el robot, en el orden correspondiente. Sin embargo, todos los algoritmos comentados anteriormente están pensados para calcular el camino óptimo de un punto, sin tener en cuenta las dimensiones físicas que pueda tener el robot. Para solucionar este problema, simplemente se ensanchan las porciones del mapa que representan obstáculos, de manera que se cree un margen de distancia entre el obstáculo y el punto central del robot (que es el punto del cual se obtienen las coordenadas de posición)

4 En [7] se discute este hecho

que corresponda con la distancia real entre el punto central y el punto de la periferia de la base. De esta manera, el problema se reduce al de guiar a un único punto a través de un camino.

Cada tramo recto consta de un conjunto de muchos puntos (con el consiguiente aumento de memoria utilizada y posible ralentización del proceso), cuanto con solo dos sería suficiente para definirlo. Para solventar este problema, se *limpia* la lista de puntos de manera que solo queden los puntos inicial y final de cada tramo recto. Para ello se verifican los puntos colineales de cada tramo y se eliminan hasta encontrar uno que no sea colineal, lo que significará que pertenece a un nuevo tramo. Se repite el proceso hasta que se llegue al punto final del camino.

Se observa que con este algoritmo, se obtiene una trayectoria formada por tramos rectos. Para tal de obtener un movimiento más natural y suave, se propone sustituir algunas de las uniones entre tramos rectos (zonas con tramos *puntiagudos*) por curvas, de manera que el movimiento sea más suave y al mismo tiempo se optimize el tiempo empleado (ya que es más rápido un movimiento curvo que parar y alinearse con el siguiente tramo). Para ello, se usan curvas de Bézier, curvando la zona de unión entre tramos, como se ve en la Figura 17.

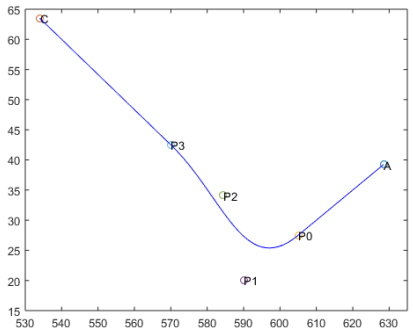


Figura 17: Curva de Bézier. Los puntos A, B y C son los que definen los tramos rectos (siendo B la intersección entre ellos). Los puntos P_0, P_1, P_2 y P_3 son los puntos de control de la curva de Bézier.

Si se encuentra un punto en el cual la curva de Bézier sale de la zona navegable, ese tramo no se deformará y seguirá siendo recto. Este proceso se repite iterativamente, repitiendo el algoritmo hasta que se llegue al punto final. Se usan curvas cúbicas de Bézier, con la condición que la curva $\mathbf{B_i}$ se una de manera suave con los tramos anterior $\mathbf{r_i}$ y siguiente $\mathbf{s_i}$ (es decir, que el vector tangente a la curva en los instantes inicial y final sean iguales a los vectores tangentes de los tramos anterior y siguiente, respectivamente. Como estos son rectas, esto equivale a decir el vector tangente a la curva en $t = 0$ y $t = 1$ sean iguales a las velocidades de los tramos anterior y siguiente, que son constantes y paralelas a las trayectorias respectivas). Se añaden como condiciones extra que la curva no tenga ningún punto con un radio de curvatura inferior a cierto valor prefijado y que la velocidad

máxima no supere en ningún momento en módulo un valor prefijado como máximo. Estas condiciones se pueden resumir como:

$$\begin{cases} \overrightarrow{B}_i(t_i) = (1-t_i)^3\mathbf{P_{0_i}} + 3t(1-t_i)^2\mathbf{P_{1_i}} + 3t_i^2(1-t_i)\mathbf{P_{2_i}} + t^3\mathbf{P_{3_i}} \\ t_i \in [0,1] \end{cases} \tag{13}$$

$$\mathbf{P_{0_i}} \in \mathbf{r_i} \tag{14}$$
$$\mathbf{P_{3_i}} \in \mathbf{s_i} \tag{15}$$
$$\overrightarrow{\mathbf{B}}_i(0) = \overrightarrow{v}_{r_i} \tag{16}$$
$$\overrightarrow{\mathbf{B}}_i(1) = \overrightarrow{v}_{s_i} \tag{17}$$
$$\overrightarrow{B}_i(t_i) \in \mathbf{C}, \forall t_i \in [0,1] \tag{18}$$
$$R_{curv}(t) > R_{min}, \forall t_i \in [0,1] \tag{19}$$
$$\|\overrightarrow{\mathbf{B}}_i(t)\| < v_{max}, \forall t_i \in [0,1] \tag{20}$$

donde el subíndice *i* representa el tramo correspondiente, t_i es un parámetro que permite definir la curva correspondiente, $\overrightarrow{\mathbf{B}_i}$ es la parametrización 2D que define la curva, $\overrightarrow{\mathbf{B}_i}$ es su derivada temporal (correspondiente al espacio tangente a la curva), **C** es el espacio de posiciones posibles dentro de la zona navegable, $R_{curv}(t)$ es el radio de curvatura local en cada instante, y $\mathbf{P_{0_i}}$, $\mathbf{P_{1_i}}$, $\mathbf{P_{2_i}}$ y $\mathbf{P_{3_i}}$ son los puntos de control de la curva correspondiente. $\mathbf{P_{0_i}}$ y $\mathbf{P_{3_i}}$ serán los puntos inicial y final de la curva en cuestión, respectivamente. $\overrightarrow{v}_{s_i}$ y $\overrightarrow{v}_{s_i}$ corresponden a los vectores de velocidad de los tramos rectos, los cuales son conocidos *a priori*, ya que el módulo de la velocidad es conocido (fijado a un valor de 0,5 m/s), así como su dirección, paralela a la recta correspondiente.

Con estas condiciones, únicamente quedan fijados los valores de dos puntos según las Ecuaciones 16 y 17. Se fijan restricciones en los valores de $\mathbf{P_{0_i}}$ y $\mathbf{P_{3_i}}$ con las Ecuaciones 14 y 15, y con las Ecuaciones 18, 19 y 20 se fijan restricciones globales de la curva. En estas condiciones, hacen falta ecuaciones adicionales para resolver el problema. Se podría buscar minimizar algún parámetro de la curva, como su longitud total o la curvatura local, y convertir el problema en optimizar la curva de acuerdo a los requisitos pedidos. Sin embargo, este planteamiento sale del alcance de este proyecto debido a su complejidad, y se utiliza en su lugar un enfoque más empírico: primeramente se expresan los puntos $\mathbf{P_{0_i}}$ y $\mathbf{P_{3_i}}$ como:

$$\begin{cases} \mathbf{P_{0_i}} = \mathbf{A} + (1-k)\overrightarrow{\mathbf{AB}} \\ \mathbf{P_{0_i}} = \mathbf{B} + k\overrightarrow{\mathbf{BC}} \\ k \in [0,0{,}5] \end{cases} \tag{21}$$

donde **A** hace referencia al punto inicial de la recta r_i, **C** es el punto final de la recta s_i y **B** es la intersección de ambas rectas. De esta manera los puntos

quedan definidos como pertenecientes a la semirrecta[5] correspondiente en función de un parámetro k[6]. Interesa que estos puntos estén lo más cerca entre ellos para que la curva sea lo más corta posible; sin embargo, no pueden estar lo cerca que se quiera, ya que entonces la forma de la curva puede verse alterada de manera innecesaria; además, entonces se consiguen giros muy bruscos. Para buscar una solución, se realiza un estudio con diferentes rectas y valores de k para ver diferentes casos y diferentes curvas posibles para un mismo caso y, a partir de ello, restringir los valores posibles de k, así como ver como evolucionan algunos parámetros de interés (como la curvatura y el módulo de la velocidad) . En el Anexo B de este proyecto puede verse este estudio. Una vez visto, el algoritmo realiza diversas iteraciones con diferentes valores de k hasta encontrar una curva que cumpla las condiciones comentadas anteriormente. Esta curva será la finalmente almacenada.

En la trayectoria final, los tramos rectos se definirán como rectas y los curvos con las curvas de Bézier descritas en Ecuación 13. Esta información se almacenará como una lista con los mínimos puntos necesarios para definir cada recta y cada tramo curvado, diferenciándolos entre ellos para tal de tratarlos posteriormente (los puntos pertenecientes a las curvas se marcaran con un parámetro extra indicativo de esto). Para definir cada tramo recto solo serán necesarios los puntos inicial y final de cada uno, y para los tramos curvos, únicamente los 4 puntos que definen la curva. Los valores de las curvas se evalúan usando la librería *bezier* de *node.js*, que permite evaluar polinomios de Bézier de cualquier grado.

En resumen, el algoritmo funciona de la siguiente manera:

- Se ejecuta el algoritmo JPS para encontrar un camino óptimo en el mapa. Este camino estará formado por el conjunto de puntos a recorrer, que forman tramos rectos.
- Se *limpia* esta lista de puntos para obtener una lista con solo los mínimos puntos necesarios para definir tramos rectos.
- En las uniones entre tramos, calcular la curva de Bézier correspondiente. Si toda ella es navegable y cumple las condiciones de curvatura y velocidad máxima, almacenar los 4 puntos necesarios para obtener la curva con un parámetro extra indicando que son puntos de una curva. Se repite este paso para todas las uniones entre tramos.
- Almacenar esta nueva lista de puntos, la cual es suficiente para indicar la trayectoria a seguir. Esta lista será luego tratada adecuadamente para hacer que el robot la siga.

5 Se eligen los segmentos a los cuales pertenece el punto correspondiente como una semirrecta de manera arbitraria para reducir el intervalo de posibles puntos, así como para evitar problemas de solapamiento de puntos si se da el caso de dos curvas consecutivas

6 Se escoge un único parámetro para facilitar las operaciones y para mover los puntos inicial y final de manera conjunta

Además, para cada curva de Bézier se calculan y almacenan las velocidades en diferentes instantes de tiempo y el tiempo que llevará recorrerla:

$$T = \sum_{n=1}^{N} \frac{s_i}{\|\vec{\dot{\mathbf{B}}}_i\|} \tag{22}$$

donde N es el número de divisiones de la curva (en este caso se escoge N = 100), s_i es la longitud de un tramo pequeño de la curva (considerado como tramo recto) y $\|\vec{\dot{\mathbf{B}}}_i\|$ es la velocidad que tiene en ese tramo (considerada constante en ese instante). Esta simplificación permite calcular la longitud de la curva de manera fácil y con un error despreciable usando un valor de N suficientemente grande). Estos datos se usarán luego para el algoritmo de seguimiento de trayectoria [7]. Así mismo, se calcula y almacena también el radio de curvatura R en cada instante dado por la expresión:

$$R(t) = \frac{(\dot{B}_x^2 + \dot{B}_y^2)^{\frac{3}{2}}}{|\dot{B}_x \ddot{B}_y - \dot{B}_y \ddot{B}_x|} \tag{23}$$

donde $\dot{B}_x$, $\dot{B}_y$, $\ddot{B}_x$, $\ddot{B}_y$ hacen referencia a las componentes correspondientes de velocidad y aceleración instantáneas, respectivamente.

El algoritmo siempre toma como posición inicial la posición actual de MASHI, y la posición final le es dada por el operador a través de la web que este tiene abierta. A través del campo de introducción de texto para TTS, el operador introduce las coordenadas de destino en el formato siguiente:

$$PATH - PLNG; X_{final}; Y_{final}$$

Se ha modificado el código fuente de la web de manera que interprete estos comandos para ejecutar el *path-finding* y no para la síntesis de voz.

5.4. Implementación de seguimiento de trayectoria

Una vez definidos los sistemas de localización, mapa y *path-finding*, queda implementar un sistema de guía, el cual permita dar lar órdenes adecuadas en cada momento a la base para tal de que esta siga la trayectoria adecuada.

La plataforma MASHI ya dispone de órdenes programadas para moverse hacia adelante, atrás, y girar hacia derecha e izquierda [8] (sin realizar movimiento de traslación). Estas funciones ya están hechas para tal de comunicarse correctamente entre operador, servidor y *Arduino*, transmitiendo y leyendo los datos necesarios para tal de efectuar el movimiento correspondiente de los motores. Como se comentó en el Capítulo 2, en este proyecto no se comenta en profundidad el funcionamiento de los diferentes códigos programados para MASHI, pero pueden consultarse todos ellos en [11]. Aprovechando este hecho, se programará en el mismo archivo *server.js* una función

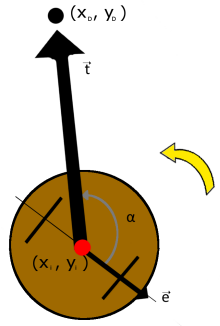
7 Estos datos se calculan y almacenan para acelerar luego la ejecución del seguimiento al no tener que calcular en tiempo real estos datos.

8 Estos movimientos siempre están referenciados en base local del robot: es decir, *adelante* se refiere a la dirección en la que el robot este mirando.

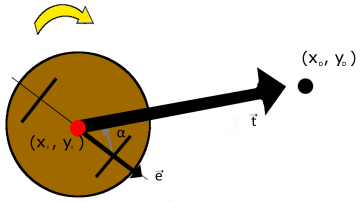
que lea las coordenadas en todo momento y de las direcciones correctas de movimiento (adelante, atrás, giro a derecha o izquierda).

La manera más intuitiva de implementar esto es haciendo que la base se oriente de manera adecuada para que, dando la orden de *adelante*, el movimiento sea siguiendo la trayectoria, la cual estará formada básicamente por líneas rectas. Es posible que pudiera el caso de tener una trayectoria con muchos segmentos, en cuyo caso este planteamiento sería inefectivo.

Así, en la Figura 18 y la Figura 19 pueden verse diversas situaciones donde se representa lo arriba descrito: el robot tiene unas coordenadas y una posición arbitrarias, y tiene que orientarse correctamente para poder seguir la trayectoria recta, o seguirla si la orientación es la adecuada.

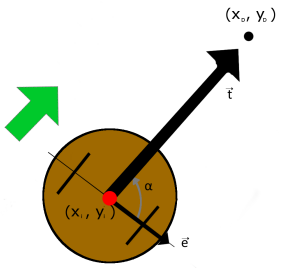


(a) El robot necesita girar a la izquierda para orientarse de manera adecuada.

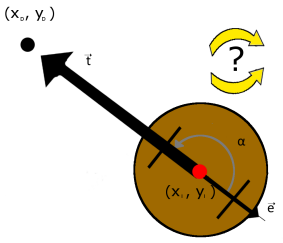


(b) El robot necesita girar a la derecha para orientarse de manera adecuada.

Figura 18: Situaciones donde el robot ha de orientarse para seguir la trayectoria.



(a) El robot tiene la orientación adecuada y yendo adelante seguirá la trayectoria.



(b) El robot puede girar tanto a la derecha como a la izquierda para orientarse adecuadamente.

Figura 19: Situaciones donde el robot ha de ir adelante y donde cualquier giro dará un resultado válido.

El primer paso de este algoritmo es calcular el ángulo entre el eje de las ruedas (equivalente al vector $\overrightarrow{e} = (1, 0)$ en referencia local del robot, como se ve en la Figura 18) y vector de traslación $\overrightarrow{t}$, que corresponde al vector

formado por el punto de destino, que se ha calculado con el algoritmo de *path-finding*) y el punto de posición actual:

$$\overrightarrow{t} = \begin{Bmatrix} x_D - x_i \\ y_D - y_i \end{Bmatrix} \tag{24}$$

donde los subíndices i representan las coordenadas actuales y D hace referencia a la siguiente posición de destino.

Estos vectores han de ser perpendiculares para que el movimiento de traslación dada con la orden de *adelante* sea en la dirección correcta, dada por el vector de traslación. Si no, la base ha de girar para orientarse adecuadamente. Para calcular el ángulo direccionado α entre $\overrightarrow{e}$ y $\overrightarrow{t}$, [9]se usa la función *atan2*, que permite obtener el ángulo entre dos vectores en el rango de 0^o a 360^o, información que ayudará a decidir si el giro ha de ser a la izquierda o a la derecha. En función de este ángulo se tomará una decisión u otra:

- $\alpha = 90^o$. Viendo la Figura 19a, se ve como esta situación es la necesaria para poder seguir la trayectoria. En este caso ha de darse la orden en dirección adelante.
- $\alpha \in (0, 180^o)$, $\neq 90^o$. Esta situación puede visualizarse en la Figura 18a. La opción más rápida en este caso es realizar un giro antihorario hasta que $\alpha = 90^o$, que corresponde con un giro hacia la izquierda.
- $\alpha \in (180^o, 360^o]$. Como se ve en la Figura 18b, esta situación es la inversa de la anterior, siendo ahora la mejor opción un giro a la derecha (en sentido horario).
- $\alpha = 180^o$. Este es un caso singular como se ve en la Figura 19b, ya que tanto un giro a la izquierda como a la derecha darán el mismo resultado.

Sin embargo, no se trabajará con valores exactos, ya que se dispone de una precisión limitada. Se usarán valores con tolerancias, de manera que el resultado al que se llegue sea igualmente válido y no haya problemas con las operaciones. De esta forma, además se evita el punto singular de $\alpha = 180^o$, ya que al usar tolerancias siempre se estará en un caso de girar a izquierda o derecha.

Este algoritmo es eficaz para conectar tramos rectos, sin embargo, aparecen dificultades al tener tramos curvos, ya que de la manera en la que están programadas las órdenes descritas, los movimientos que se pueden realizar son secuenciales y no pueden ser simultáneos (el robot no puede moverse en direcciones arbitrarias combinando movimientos). Para estos casos, se propone una generalización del algoritmo propuesto, en el cual se calcula, a partir de la ecuación de la curva, la velocidad necesaria de los motores para tal de efectuar esa trayectoria. Al disponer de la ecuación en forma explícita y saber *a priori* su forma (un polinomio de Bézier de tercer grado), es inmediato calcular la derivada de la función, la cual permite calcular la velocidad necesaria, y a partir de esta obtener la velocidad de giro de las ruedas,

9 Este ángulo hace referencia al ángulo que va desde $\overrightarrow{e}$ hacia $\overrightarrow{t}$, con el sentido de giro correspondiente.

siguiendo un método similar al sistema de odometría discutido en la Sección 5.1, pero de manera inversa, ya que a partir de valores de velocidad en base global se calcularán valores de velocidad angular de ambas ruedas[10]:

$$\overrightarrow{v}(t) = \frac{R}{2}(\dot{\varphi}_R - \dot{\varphi}_L)\begin{Bmatrix}\cos(\theta)\\ \sin(\theta)\end{Bmatrix} = \overrightarrow{\dot{B}}(t) \tag{25}$$

Para eliminar la incógnita θ se trabaja con el módulo:

$$\|\overrightarrow{\dot{B}}(t)\| = \frac{R}{2}(\dot{\varphi}_R - \dot{\varphi}_L) \tag{26}$$

Con esta ecuación puede calcularse la diferencia entre las velocidades angulares de los motores, pero hace falta una expresión adicional para calcular los valores concretos. Para ello se utiliza la Ecuación 7, y teniendo en cuenta que a cada instante, la distancia al CIR corresponderá con el radio de curvatura *r*, el cual se ha calculado en la Ecuación 23 y almacenado previamente junto con las velocidades de la curva.

Combinando la Ecuación 23 y la Ecuación 26 se obtiene:

$$\begin{cases}\dot{\varphi}_L = \frac{2\sqrt{\dot{B}_x^2+\dot{B}_y^2}}{R}\left(\frac{R_C}{R_C+L}-1\right)\\ \dot{\varphi}_R = \frac{r\dot{\varphi}_L}{L+R_C}\end{cases} \tag{27}$$

donde *R* es el radio de las ruedas, *L* la distancia entre ellas, R_C es el radio de curvatura local y $\dot{B}_x$ y $\dot{B}_y$ son las componentes del vector velocidad. De esta ecuación se puede derivar la implementación comentada anteriormente para seguimiento de tramos rectos, ya que es un caso particular aprovechando las órdenes ya programadas en MASHI.

La curva de Bézier está parametrizada por un valor *t* que no coincide con un valor temporal *real*, así que se realiza un cambio de variable para poder determinar en cada instante la velocidad necesaria. Para ello, se utiliza el tiempo total de recorrido T calculado en la Ecuación 22 para hacer el siguiente cambio:

$$\bar{t} = tT \tag{28}$$

A partir del valor de $\bar{t}$ (correspondiente al tiempo *real* transcurrido desde el inicio del seguimiento de la curva), se calcula el valor correspondiente de *t* y se selecciona el valor de velocidad de la lista calculada en la Sección 5.3. Este proceso se repite hasta que el tiempo $\bar{t}$ sea igual o superior al calculado, momento en el cual la curva ha finalizado y comienza el nuevo tramo recto.

El algoritmo resultante es una combinación del algoritmo para tramos rectos y para tramos curvos:

- La plataforma se orienta correctamente para seguir el primer tramo recto de su camino.

10 Estas velocidades son función del tiempo, pero por comodidad no se expresa explícitamente como ello.

- Se sigue el camino hasta encontrar el primer punto de una curva o bien el primero punto de giro (el cual solo estará si no se ha podido encontrar una curva de Bézier en ese tramo).
- Si el punto es perteneciente a una curva, se dan las órdenes necesaria de velocidad, calculando las velocidades motoras correspondientes a cada instante; si no, se ejecuta el algoritmo arriba descrito para orientarse correctamente.
- Se recorre el siguiente tramo recto, y se repite el proceso hasta llegar al punto final.

Para implementar esta función, se programa una función en el archivo *server.js* que vaya leyendo las coordenadas a cada momento, calcule el vector $\overrightarrow{t}$ usando la posición actual y el siguiente punto de destino, el vector $\overrightarrow{e}$ en base global, calcule el ángulo entre ambos y en función de ello avance, gire a la derecha o a la izquierda. Cuando el módulo del vector $\overrightarrow{t}$ sea nulo (o tenga un valor muy pequeño al trabajar con tolerancias), querrá decir que se ha alcanzado el punto objetivo, y se elimina de la lista de puntos el primero (que corresponde al punto de destino ya alcanzado). Si el punto actual es un punto marcado como punto de Bézier, se calculan las velocidades angulares necesarias como se indica en la Ecuación 25 y se envían al Arduino para que este mueva los motores de la forma requerida. El algoritmo trabaja en bucle hasta que la lista de puntos de destino este vacía (es decir, que el punto de posición actual corresponda con el punto final de la lista, que es el punto objetivo).

6 Pruebas con MASHI

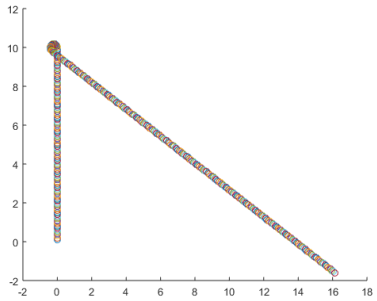
Para comprobar la funcionalidad de los algoritmos descritos en el Capítulo 5 y de la plataforma en sí, se realizaron diversas pruebas. Además, durante una semana se recuperaron los elementos originales de MASHI para llevarlo a la *Smart City Expo World Congress 2016* celebrada en Barcelona, donde fue expuesto y pudo verse en directo su funcionamiento. Acabado el congreso, los componentes originales fueron devueltos a la ESPOL. Justo antes del congreso, los *encoders* de los motores de la base fallaron, y como se aprecia en el Capítulo 8, una gran parte del tiempo posterior al congreso fue utilizado en identificar y solucionar el problema. Para ello, se ha cambiado el sistema de movimiento de la base pasando a ser en lazo abierto (sin utilizar el PID), ya que a la hora de identificar el problema ya no quedaba suficiente tiempo para comprar e instalar un nuevo *encoder*, por lo que se optó por la solución más rápida. Cambiando adecuadamente el código de *Arduino*, se pudo conseguir que pudiera trabajar en lazo abierto.

6.1. Pruebas de localización

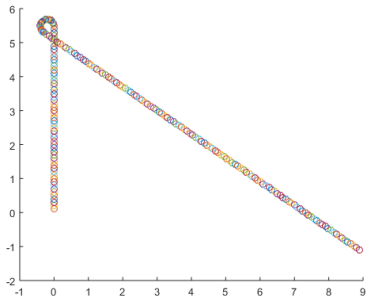
El sistema de odometría se probó en el laboratorio, moviendo la plataforma de manera manual con las órdenes preprogramadas y mostrando en la consola del PC los resultados de los cálculos. Al tener un sistema de lazo abierto, no se puede leer la velocidad de cada motor, y se ha tenido que suponer que la velocidad real de cada motor como igual a la velocidad ordenada. Al estar cambiado el sistema a lazo abierto y sin ningún tipo de control, estas velocidades no son iguales en la realidad (debido seguramente a que el sistema tendrá una ganancia inferior a la unidad). Sumado al hecho que el sistema de odometría no tiene una elevada precisión (con un error que tiene a divergir), el sistema de localización discutido en este proyecto en las condiciones actuales no es viable. Este hecho se pudo comprobar en el laboratorio, moviendo el robot en línea recta se obtuvieron resultados imprecisos. El ángulo girado también se calculó de manera imprecisa, y al ser una magnitud más sensible que las otras, este error influye de manera más notable. En el siguiente capítulo se comentan las posibles soluciones a este problema. Para comprobar si el sistema de odometría es válido o no, se realizó una simulación en *MATLAB* con una serie de velocidades de los motores programadas (con un error incluido), graficando las coordenadas (en metros). En la Figura 20 se pueden ver estos resultados.

6.2. Pruebas de *path-finding*

Para las pruebas de *path-finding* se ejecutó el algoritmo descrito en la Sección 5.3 con el mapa de prueba de la Figura 16. Se obtuvo la trayectoria de



(a) Trayectoria ideal (sin error de lectura de velocidad).



(b) Trayectoria con error de lectura de velocidad (igual a la centésima parte de un valor aleatorio normal estándar).

Figura 20: Simulaciones del sistema de odometría, ideal y con error. Se ve como un error muy pequeño en la lectura de la posición causa errores catastróficos en los cálculos de posición. Se puede concluir que el sistema de odometría por sí solo no es capaz de dar una solución robusta al problema de localización.

la Figura 21 . Se puede observar como es un camino formado por tramos rectos y curvos que cumplen las condiciones mencionadas en la Sección 5.3.

6.3. Pruebas de seguimiento de trayectoria

Al no disponer de un sistema robusto de localización, se realizó una modificación del algoritmo descrito en la Sección 5.4 para que se simulen las coordenadas en función de la orden dada; de esta manera se simula el movimiento como si se tuviese un sistema de localización funcional y preciso. Luego se ejecutó el algoritmo y se graficó la trayectoria obtenida simulada, obteniendo el camino de la Figura 22.

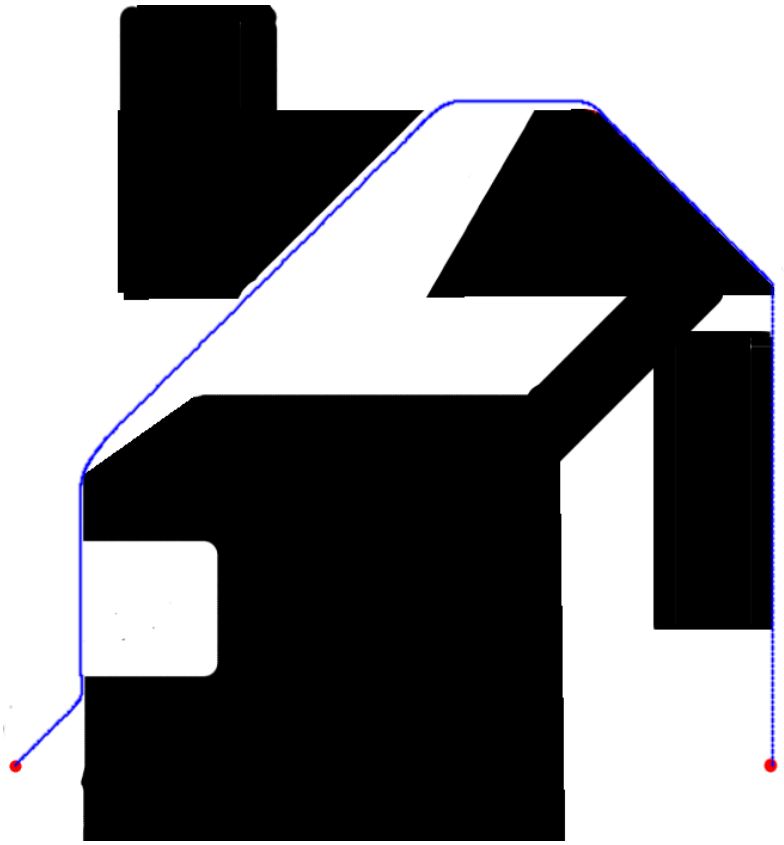


Figura 21: Trayectoria obtenida por el algoritmo descrito en la Sección 5.3 con los puntos marcados en rojo como inicial (izquierda) y final (derecha). Se aprecian las zonas curvadas.

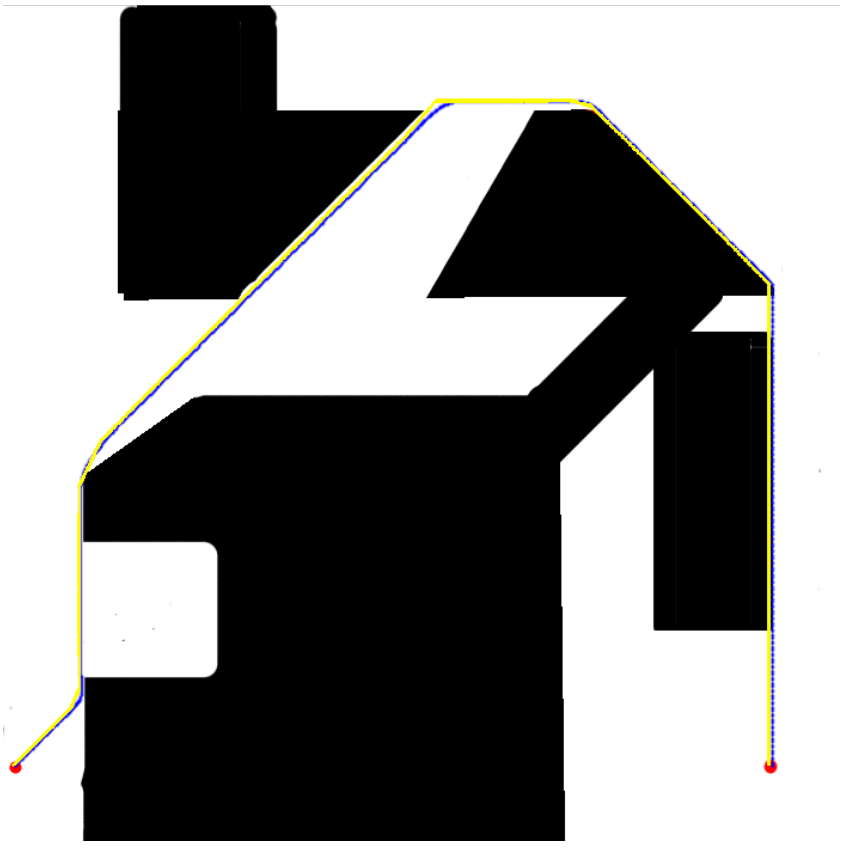


Figura 22: Trayectoria obtenida por la simulación (en amarillo). Se ve como co con la trayectoria calculada previamente con muy poco margen de

7 Posibles mejoras. Trabajo futuro

El principal objetivo de este proyecto era reconstruir la plataforma MASHI original, objetivo que se ha cumplido satisfactoriamente. Se ha añadido una mejora en la navegación también, enriqueciendo su funcionalidad. La plataforma MASHI, sin embargo, tiene muchos aspectos a mejorar, y las mismas mejoras implementadas pueden optimizarse mucho más, ya lo que se buscaba en este proyecto era más bien construir un *framework* de trabajo sobre el cual posteriores proyectos puedan tener una base sólida sobre la cual empezar y desarrollar temas.

La reconstrucción de MASHI ha ocupado la mayor parte del tiempo del proyecto, así que la mejora de navegación no ha podido desarrollarse de la manera más óptima y eficaz posible. Además, debido al fallo de los *encoders* de los motores, el sistema ha tenido que usarse en lazo abierto, perdiendo precisión y optimalidad: la mejora propuesta en este proyecto necesita uso de lecturas de velocidad, así que su implementación no ha sido del todo correcta a nivel práctico. La sustitución de estos sensores es prioritaria para futuros trabajos.

Los subsistemas desarrollados podrían optimizarse para trabajar de manera más eficiente, o bien cambiarse del todo para mejorar su funcionalidad:

- El sistema de localización basado en odometría tiene el principal inconveniente que tiende a divergir con el tiempo, motivo por el cual su precisión disminuye. Se comprobó en la Sección 6.1 como este método de localización por si solo no es válido para este proyecto. La solución a este problema es mejorar el sistema de posicionamiento mediante la adición de una IMU o bien de un sistema de triangulación/-trilateración como los comentados en la Sección 4.3. Estos elementos se combinarían con la odometría mediante algún sistema de filtraje (preferiblemente un filtro de Kalman) para obtener una localización precisa.

- El sistema de mapa es bastante simple, aunque funcional. Podría optimizarse almacenando únicamente las zonas navegables mediante objetos matemáticos más generales que una matriz de todos los posibles puntos (como por ejemplo, almacenando los polígonos que definen las zonas navegables). De esta forma además podría obtenerse un mapa continuo en lugar de discreto si así se quisiese.

- El sistema de *path-finding* utilizado encuentra el camino más corto, pero no necesariamente el más rápido: puede darse el caso de tener un camino formado por muchos tramos rectos de corta longitud (como se ve en la Figura 23) y que no puedan formarse curvas como las descritas en la Sección 5.3. En este caso, el algoritmo de este proyecto para el seguimiento de trayectoria resultaría ineficaz. Una solución seria adaptar

un sistema como el comentado en [3], donde se obtiene una única trayectoria curva. Otra opción sería modificar el algoritmo de *path-finding* para tener en cuenta el coste temporal (teniendo en cuenta el tiempo de giro) en lugar del coste de distancia.

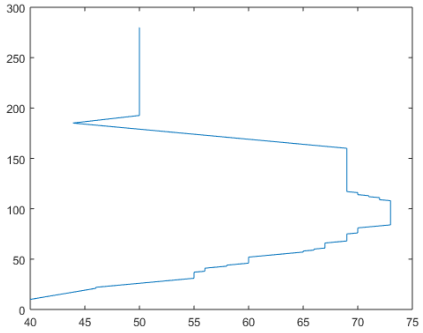


Figura 23: Ejemplo de camino *escalonado* formado por tramos cortos. En este tipo de camino el algoritmo descrito resulta ineficaz debido al tiempo requerido en estar parando y girando continuamente.

- El algoritmo de seguimiento de trayectoria podría generalizarse para cualquier tipo de camino, en lugar de diferenciar entre tramos rectos y curvos y aplicar un algoritmo u otro en cada caso. Esto se ha hecho por conveniencia para aprovechar las órdenes ya programadas de antemano en MASHI.

Así mismo, la misma plataforma MASHI puede mejorarse y optimizarse en más aspectos que han quedado fuera de este proyecto: al ser una plataforma experimental, muchas partes del diseño han sido hechas de manera más funcional que óptima. Por ejemplo, el diseño mecánico de las partes móviles puede mejorarse para conseguir mejores movimientos y reducir vibraciones; el diseño estructural también puede estudiarse más a fondo para encontrar materiales y formas más eficientes para la estructura del robot; el diseño eléctrico y electrónico también puede simplificarse y mejorarse, ya que los componentes actualmente están sobre una placa de prototipado con un *Arduino* y, al tener un diseño definitivo, se podría optar por usar un circuito impreso con los componentes adecuados para cumplir la misma función, ahorrando en espacio y en costes; finalmente, toda la parte de código también podría optimizarse, ya que aunque no se requieren gran cantidad de recursos para ejecutarlo, existen muchas líneas y funciones *basura* dentro del código, haciendo falta una depuración de todo el código. La definición de los posibles movimientos de MASHI podría mejorarse en gran medida, ya que no resulta sencillo moverlo de manera precisa.

8 Programación temporal

En la Figura 24 puede verse un diagrama de Gantt del proyecto. Se puede ver como la reconstrucción ha abarcado la mayor parte del tiempo disponible. Así mismo, se han intentado desarrollar varias tareas en paralelo (como por ejemplo tanto el estudio de la plataforma y sus subsistemas como su reconstrucción física). Al no disponer de documentación previa, esta tarea ha llevado un trabajo mas exhaustivo que si se hubiese dispuesto de ella. Así mismo, como se comentó en el Capítulo 6, a mediados de noviembre MAS-HI fue expuesto en la *SCEWC*, semana durante la cual no se pudo trabajar a nivel físico sobre el robot, pero pudo verse en directo su funcionamiento. Las semanas 12, 13 y 14 la impresora 3D estuvo fuera de servicio, por lo cual se llevó a reparar y no puedo imprimirse en ese periodo.

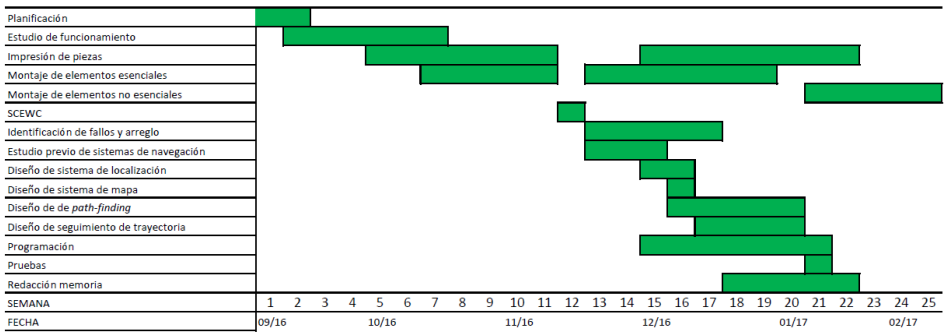


Figura 24: Diagrama de Gantt del proyecto

9 Presupuesto

A continuación de muestra un presupuesto de los gastos del proyecto y lo que supone desarrollar la plataforma MASHI:

-Componentes:

- Robotis OpenCM 9.04A 20 €
- Display AT070TN90 75 €
- Altavoces Energy Sistem Music Box Z30 30 €
- 1 perfil aluminio 15x15x400 mm 5 €
- 2 x varillas PVC ∅ 5mmx x 2m 5 €
- 3 x rollos PLA 90 €
- Tornillería diversa 10 €
- Espuma protectora 5 €
- Parallax *Motor Mount and Wheel Kit* 300 €
- 2 x Parallax HB-25 100 €
- Batería 12V DC 30 €
- Arduino Mega 2560 40 €
- Placa Protoboard 5 €
- 2 x Webcam 40 €
- 6 x Dynamixel AX-12A 408 €
- Base madera ∅ 400 mm 20 €
- Cilindro metálico ∅ 50 mm x 1100 mm 20 €
- Juego de engranajes VEX 25 €
- Varillas VEX 20 €
- Collar de eje VEX 13 €
- Cableado eléctrico 10 €
- Pegamento 10 €
- Cables TTL 20 €

TOTAL: 1301 €

-Recursos humanos:

- 25 semanas x 20 h x 20 €/h 10.000 €

-Costes amortizables:

- Impresora 3D amortizable en 5 años 221 €
- PC portátil amortizable en 3 años 64 €

-Contingencia:

- 10 % del subtotal 1158 €

TOTAL: 12.744 €

10 Estudio medioambiental

Para conocer el impacto medioambiental de este proyecto, se calcula el gasto energético producido por el uso de MASHI. Primeramente se calcula cuanto consume MASHI en estado operativo (suponiendo que esta la mitad de su tiempo de funcionamiento moviéndose y la otra mitad estático). Estos resultados pueden verse en la Figura 25.

Componentes	P (W)	Cantidad (uds)
Motor Parallax+encoder	18,25	2
HB-25	1	2
Motor Dynamixel	10,8	6
Arduino Mega	0,25	1
Robotis Open CM	0,2	1
Display AT070TN90	7	1
Altavoces	3,9	1
Camara	1	2
PC	70	1
Potencia en movimiento	186,65	
Potencia estática	150,15	
Potencia total	168,4	
Energia en 1h de uso (kWh)	0,1684	

Figura 25: Consumo de los componentes de MASHI

Esta energía se compara con el consumo medio de una persona haciendo la misma función de MASHI, como se ve en la Figura 26 para tener una comparativa y ver si la telepresencia de MASHI supone un ahorro energético [1] [2].

MASHI	0,1684 kWh
Humano caminando	0,12312292 kWh
Humano operador	0,096 kWh
Humano en funciones de MASHI	0,108784 kWh
MASHI+operador	0,2644 kWh
Coche	35,5 kWh

Figura 26: Comparativa de consumos medios de MASHI: una persona caminando, una persona haciendo la función de MASHI (30 min caminando y 30 min quieto), una persona quieta haciendo de operador y un coche.

1 Se ha calculado el consumo energético humano según la fórmula de Harris-Benedict presente en [2], usando una persona de 70 kg, 180 cm de altura y 40 años de edad

2 Para el consumo del coche se ha usado el consumo de un Audi A3 MY17 1.4 TSI AUT. 7V 150CV PK35 viajando a 80 km/h

Se puede ver que la plataforma MASHI tiene un consumo mayor que una persona haciendo la misma función; sin embargo, hay que tener en cuenta que la persona ha de desplazarse hasta el lugar de trabajo, y este desplazamiento supone un gasto energético. Así, si el desplazamiento se hace a pie, se ve que a partir de un poco más de 1h caminando, el consumo total humano ya supera al consumo de MASHI, así que si se tiene que emplear más tiempo de ese para el desplazamiento a pie, la plataforma MASHI ya sale rentable desde el punto de vista energético. Si el transporte se realiza en coche, el consumo se dispara, por lo que el uso de MASHI permite ahorrar esta energía, ya que la telepresencia evita la necesidad de realizar este desplazamiento.

11 Conclusiones

En este proyecto se ha podido ver como funciona una plataforma de telepresencia, entendiendo su funcionamiento a nivel técnico. Se ha reconstruido la plataforma MASHI con sus funcionalidades originales y se ha documentado el proceso tanto de reconstrucción como de descripción del robot, creando así una referencia para posteriores trabajos y creando un *framework* para facilitar futuras investigaciones sobre la plataforma. De esta manera, la plataforma MASHI vuelve a ser operativa tanto para uso experimental como para un posible uso real (haciendo acto de telepresencia). Su estancia en la *SCEWC 2016* ayudará a ampliar el interés por este tipo de tecnologías, facilitando su posterior desarrollo.

Además, se ha desarrollado e implementado una función de *path-finding* en el robot para buscar y seguir una trayectoria curva y óptima. Para ello se desarrollaron sistemas de localización, de cálculo de camino más corto y de seguimiento de trayectoria. Aunque un fallo de los sensores del robot no ha permitido implementar al 100 % estas funciones, las simulaciones han verificado que el sistema de cálculo de trayectoria funciona correctamente y el resultado es válido. El sistema de localización no ha resultado ser viable, ya que los errores hacen que el sistema no sea robusto. Finalmente, el sistema de seguimiento de trayectoria ha resultado válido en simulación, dando como resultado un camino seguido prácticamente idéntico al calculado. El pequeño error puede solventarse añadiendo un sistema PID para reducir el error del sistema o simplemente añadiendo un margen de error de distancia en las zonas no navegables a la hora de dibujar el mapa.

Futuros proyectos pueden optimizar los algoritmos aquí descritos para tal de reducir su uso de memoria, mejorar la precisión y la estabilidad de los sistemas, o similares. Será prioridad en todo caso tener un sistema de lazo cerrado con un sistema preciso de localización para tal de poder obtener buenos resultados.

Se ha visto también como las plataformas de telepresencia ayudan al ahorro del consumo energético al evitar tener que realizar un desplazamiento físico de una persona, con toda la energía que ello requiere.

12 Bibliografía

[1] J. Agulló. *Mecànica de la partícula i del sòlid rígid*. Publicacions OK Punt, 2002.

[2] J. Arthur Harris and F. G. Benedict. A biometric study of human basal metabolism. In *Proceedings of the National Academy of Sciences*, volume 4, pages 370–373, 1918.

[3] J. Choi, R. Curry, and G. Elkaim. Piecewise bezier curves path planning with continuous curvature constraint for autonomous driving. In *Machine Learning and Systems Engineering*. Springer Science + Business Media, 2010.

[4] D. Harabor and A. Grastien. The JPS pathfinding system. In *Fifth Annual Symposium on Combinatorial Search*, 2012.

[5] W. Hereman and W.S. Murphy Jr. Determination of a position in three dimensions using trilateration and approximate distances. *Department of Mathematical and Computer Sciences, Colorado School of Mines*, 1995.

[6] S. Koenig and M. Likhachev. D* lite. *AAAI-02*, 2002.

[7] A. Mackworth. Iterative Deepening and IDA*, 2013.

[8] Hyung Gi Min and Eun Tae Jeung. Complementary filter design for angle estimation using MEMS accelerometer and gyroscope. *Department of Control and Instrumentation, Changwon National University*, pages 4–8, 2015.

[9] OCCC. Nota informativa sobre la metodologia de estimación del mix eléctrico por parte de la oficina catalana del cambio climático, 2016.

[10] D. Paillacho, C. Angulo, and M. Díaz. An exploratory study of group-robot social interactions in a cultural center. *IROS 2015 Workshop Designing and Evaluating Social Robots for Public Settings*, pages 44–48, 2015.

[11] D. Paillacho, J. Cortés, and Rodríguez X. MASHI code. https://github.com/jcf9410/MASHI-code, 2016.

[12] X. Rodríguez. Rediscovering the experimental robotic platform MASHI. Trabajo final de grado, Universitat Politècnica de Catalunya, 2017.

[13] J. Rosell. Planning & programming. Motion planning, 2012.

[14] G. Welch and G. Bishop. An introduction to the Kalman filter. In *SIGGRAPH*, 2001.

[15] X. Xu. Pathfinding.js. https://github.com/qiao/PathFinding.js, 2011-2012.